6.1 Recap

1. **Sequence Alignment**
   (a) **Global Alignment** Needleman-Wunsch Algorithm (1970)
   (b) **Local Alignment** Smith-Waterman Algorithm (1981)
   (c) **Gapped Alignment**

- **Computer Science:**
  - Dynamic Programming
  - Backtracking
  - Graph Theory
  - Longest Path in Directed Acyclic Graph (DAG)
  - Global-Local
  - Genome Alignment
- **Statistics:**
  - Heuristic Interpretation of Alignment Score as Likelihood.
  - Similarity Matrices: Entries represent statistical hypothesis testing.
  - “Denoising:” How long of a sequence do we need to view? What is “nonrandomness?” How much information (bits) do we need to infer biological significance?
- **Biology and Evolution**
  - Similarity Matrix
  - Evolutionary Model
  - Markov Model (PAM)

6.2 More on Gapped Alignment

Informally, we mean multiple things when we use the term “gap”. When speaking more formally, we can refer to each individual base that is aligned to a gap character as a gap, and each contiguous region of gaps as a gap cluster. Consult Figure 6.1 for an illustration of the above.

We can interpret the significance of affine gap alignment in this light: here the similarity score of an alignment is $\alpha$ for each match, $-\beta$ for each mismatch, $-\gamma$ for each gap cluster, and $-\tau$ for each gap.
In the above alignment, there are 4 gap clusters, and 12 gaps.

**Figure 6.1: Gaps and Gap Clusters**

![Gaps and Gap Clusters](image)

**Figure 6.2: Edit Graph for Local Alignment**

**6.2.1 Biological Significance of Alignment**

At a high level, insertions and deletions occur in natural DNA sequences, so tolerating these in alignments is necessary. Additionally, sometimes mutations change one letter to another, requiring us to tolerate mismatches as well. Sometimes, similar errors occur due to technological errors in DNA sequencing technology.

As discussed last lecture, affine gap penalty alignment tolerates a small number of large gaps (relative to linear gapped alignment), which enables us to solve problems like aligning cDNA to a reference genome.

**6.3 More on Graph Theoretical Alignment**

In Lectures 2 & 3, we saw edit graphs for global alignment of 2 and 3 sequences. Here an edit graph is given for local alignment in Figure 6.2.

This edit graph differs from that of global alignment in 2 key ways: each cell in the matrix has an edge from $V_{0,0}$ and a cell to $V_{n,m}$. 
The first edge represents the 0 term in the maximum of the local alignment recurrence: when this term is taken, then a local alignment starts at the current cell. In the edit graph, this is represented by an edge from $V_{0,0}$, which essentially short-circuits all previous cells in the alignment. Similarly, the second edge new represents finding the optimal prefix: this is the case where we traverse the Dynamic Programming matrix to find the optimal cell. In the edit graph, we see that this edge short-circuits anything following a particular cell.

Together, these edges allow an alignment to “jump in” to any cell of the matrix, gap, match, and mismatch as is fit, and at any point “jump out” and terminate the alignment.

6.4 Backpointers and Backtraces

The pseudocode presented in class for global and local alignment is focused on calculating the score of the optimal alignment, but obviously for many applications, we also want to know what the optimal alignment is.

There are several strategies for obtaining the optimal alignment in software. This discussion applies to global alignment, but may easily be generalized to local alignment and gapped alignment as well.

The simplest thing to do is probably to fill out the Dynamic Programming matrix, then starting at the bottom right cell of the matrix, and recalculate the max that lead to the cell’s value. This time, rather than just taking the max, record the cell from which the max originated (resolving ties arbitrarily: recall that optimal alignments aren’t in general unique). In other words, in the general case of global alignment, determine whether the term of the recurrence originating from an $x$-gap, a $y$-gap, or a match/mismatch was optimal, and backtrace to the corresponding position.

To generalize this technique to local alignment, we need to start at the highest-valued cell in the Dynamic Programming matrix, and cease once we reach a cell with value 0. To generalize to affine gap alignment, the same logic applies, but it becomes more complicated as there are many more cases. We first examine $V_{i,j}$, and determine whether the maximum term selected $G_{i,j}$, $F_{i,j}$, or $E_{i,j}$. From there, we determine where the value in the maximal cell originated, and backtrace accordingly.

The above strategy is in some ways simple, because it does not require us to store any extra information, but it is in some sense repetitive, as we need to go back through the matrix and recalculate how each cell along the alignment’s value was derived. With this as motivation, we now consider the alternative strategy: backpointers.

Suppose in addition to maintaining the score matrix (or matrices), as in the pseudocode we have seen thus far, we also maintain a matrix consisting of backpointers. A backpointer can be visualized as an arrow, and it simply represents where any optimal alignment that passes through a given cell came from. Backpointers are filled out at the same time as the score matrix (matrices), and they ensure that we don’t need to repeat any work when backtracing to find the optimal alignment. Instead of the above strategy, all we need do is move to the cell pointed to by each backpointer.

As far as implementations go, a backpointer can be anything that gives sufficient information for you to determine where the optimal alignments running through a cell came from. A simple strategy is to use a single integer, representing an $x$-gap, $y$-gap, or match/mismatch, and a more general strategy is to use a pair of indices, representing the cell from which a given cell’s optimal alignment came from. This strategy can be generalized to also work in gapped alignment, if we also maintain from which of the four Dynamic Programming matrices each backtrace originates.